L.P.C.B.

Peersoft v1.5

An Applesoft extension

Benoît GILON

28/02/2013

Peersoft is an Applesoft extension which focuses mainly on performance issues rather than features. Currently, it is to be considered as a complement to the Bananasoft utility already released attempting, among other features, to enrich the Applesoft interpreter by adding new keywords found in other flavors of Microsoft Basic.

An introduction to Peersoft

Peersoft is an Applesoft BASIC extension which aims to provide to the Applesoft programmer features in addition to those offered by plain vanilla Applesoft interpreter alone. Currently, in its present incarnation, Peersoft runs on DOS 3.3 only but a ProDOS version could see the day in months to come if demand level is high enough.

However, three versions of Peersoft exist depending on the CPU detected within the Apple 2 host (either 6502, 65C02 or 65802/16) with relevant code optimizations for the two latter CPUs.

The focus has been put on performance aspects while authoring this software. Here are the main features delivered with this release of Peersoft:

- New instructions for defining a default variable type per name's first character:

Tableau 1: New statements for specifying a variable's default type

New statement	Type of variable	Magnitude	Suffix character
DEFBYTE	8bits integer	From -128 to 127	.
DEFINT	16bits integer	From -32768 to 32767	%
DEFSNG	Floating point (5 bytes)		!
DEFSTR	String	Length from 0 to 255	$
DEFLNG24	24bits integer	From -8388608 to 8388607	[
DEFLNG32	32bits integer	From -2147483648 to 2147483647	]

  Therefore the program text can be smaller in size by using default typed variables.
  Using new integer types can make array variables use less memory than plain 16bits integer or FP arrays. The support for new integer types means that:

  - Some utilities which worked great for plain vanilla Applesoft and its support for the four kind of variables (integer, FP, string and user defined functions) might not work anymore with Peersoft given the fact that the way to discriminate integer variables is based upon examining the value of the `INTTYP` page zero location after a call to `PTRGET` subroutine ($80 for 16bits integer type, $81 for 8bits integer type, $83 for 24bits integer type and $84 for 32bits integer type);
  - New subroutines are provided for direct arithmetic operations implied by the four +=, -=, *= and /= operators (see below for introduction to those new statements);
  - The three new integer subtypes are now allowed for types of integer variables used within FOR/NEXT loops (with native arithmetic operations within NEXT statement handling).

- A new syntax scheme for setting variable values based upon the result from an arithmetic operation.
  `LET <variableName> += <value>` or
  `LET <variableName> /= <value>` or
  `LET <variableName> -= <value>` or
  `LET <variableName> *= <value>`
  If specified variable is of type integer, then used arithmetic is of type integer too, also the `+=` syntax scheme can be applied to string variables as well. The main benefit, beyond using integer arithmetic when adequate, is to minimize the number of variables references

compared to a syntax as in the statement `LET <variableName> = <variableName> <operator> <value>` particularly when *variableName* is a multi-dimensional array with index values equals.

- Before the release of Peersoft v1.5, an array element from array named `A` could only be referred to by using the expression `A(<expression>, <expression>)` presuming that the array A was defined as having two dimensions (with a `DIM` statement), this is the common way of using array variables from within the program text.
  But now with Peersoft v1.5, the same element could be referred to by using only one dimension specification. The two code extracts below:
  ```
  10 DIM A(4,5):…:S = 0: FOR I = 0 TO 4: FOR J = 0 TO 5:S += A(I,J): NEXT J,I
  10 DIM A(4,5):…:S = 0: FOR I = 0 TO 29:S += A(I): NEXT I
  ```
  Will both compute the sum of every element value into the S variable. This new way of authoring code will simplify some computations upon arrays and thus enhance performance of such computations.
  Also within the original Applesoft interpreter, there was a limitation that an expression value for giving a dimension must be less than 32768. At the time Applesoft was authored this was sensible as the smallest size element type was integer and takes 2 bytes for storage, given the hardware limitation of 64K for 8bits architectures: now this limitation was increased to 65535 thus addressing up to 65536 1 byte elements (such as bytes) within 64Kb.
- A new pseudo variable ("`@`") , usable in arithmetic expressions and which replicates the value currently stored in the Applesoft floating point accumulator. For instance, the statement:
  ```
  S = 0
  FOR I = 0 TO 9: FOR J = 0 TO 9:S += A(I,J) *  @
  NEXT J,I
  ```
  will compute the sum of squared elements from matrix A. You will notice that the code only refer to every element of matrix A just once and that the * operator between a value and itself is known to be (and actually is) faster than the concurrent expression `A(I,J) ^ 2`.
- Integer variables now allowed as loop variables of FOR/NEXT loops, also arithmetic operations/comparisons when running a NEXT statements is of integer type whenever the loop variable is of type integer itself.
- Fixes to some of the Applesoft instructions processing (`ONERR, RETURN and POP`): if you get a look at the Applesoft disassembly listing generated by S-C Documentor and commented by Bob Sander-Cederlof (http://www.txbobsc.com/scsc/scdocumentor/), then you'll find out that Applesoft has many bugs buried in its code.
- Utility routines provided to optimize access to Applesoft variables. For either simple or array variable kinds, time spent to get a reference to a variable is proportional to the number of variables of same kind (i.e. simple or array) that have been defined (i.e. referred to from BASIC program text for simple variables) before the time the current variable (looked up for) has been created. The order of definition of variables within the program flow seldom follows an order with which most used referred to variables or arrays are created first. We tend to define some "constant variables" such as `D$ =  CHR$( 4)` at the very beginning albeit the fact that such variable will be much less frequently used than, for instance a loop variable from an inner loop. The goal of the provided utility routines is to **optimize** variable

access **after** variables **have been created** and thus their memory storage address defined. Two alternative approaches have been supplied:
    - Physical re-organization of array/simple variables areas where actual variable slot are actually moved within the area;
    - Cache based variable referencing: here the variables storage slots are kept at the same original place within memory. Instead, Peersoft manages a small memory area to keep some variable names and addresses in a safe place. Such variables are looked up for at first place when a reference to some variable need to be returned and thus lookups for such variables are the fastest.
- Availability of co-routines within an Applesoft program: this is a new paradigm for every Applesoft program author. Now he is able to design his programs **as if** they run under a multi tasking environment. The application could be considered as an assembly of phases where the kernel is active (co-routines active and running concurrently) and where kernel is inactive (the kernel is inactive: only one flow is processed by the Applesoft interpreter). Beyond what follows, an subsequent entire chapter has been dedicated to describe its configuration and operation within this document.
    - Because the switch between co-routines will occur at known and arbitrary locations within the interpreter loop, then the context can be kept rather small compared to true multi tasking monitors.
    - Unlike true multi tasking monitors, no hardware generated external signal (IRQ) is used and thus Peersoft co-routines run equally well on the whole Apple 2 range (from the Apple II standard with Autostart ROM to Apple //gs and every emulator environment I currently know of).

Peersoft roadmap

The features described in the previous section are actually implemented in the Peersoft current release. However, not all the features envisioned from the start have already been implemented. Some are yet to be developed from scratch in order to meet the author's original requirements.

The table below provides some hints about what additional features will be developed and candidate release dates. However, as this project is founded upon the time left only as I am "idle" on both the other (i.e. family and professional/business) aspects of my life, I would suggest to not hold your breath.

Version	Feature(s) implemented	Estimated delivery date
1.5	Generalized user defined functions (`DEF FN`): more args allowed and arg. Variables of any type, not just FP)	31/04/2013
1.5	Support for remaining utility routines to reorganize the Applesoft variables areas (both simple and array)	31/04/2013
1.6	Merge with Bananasoft utility (using similar technology but focusing on features rather than performance); the name of resulting software could be "fruit salad" but still quite unsure about this ;-)	08/09/2013
1.6	Generalized integer arithmetic for expression evaluation (i.e. the sub expression A% + 1 will be evaluated using integer arithmetic first and only reverting to FP operation in case an overflow occurs	08/09/2013

Version	Feature(s) implemented	Estimated delivery date
1.7	Compilation of Applesoft user defined functions to convert them to machine code callable with `USR<n>(…)` functions (provided by Bananasoft, n from 0 to 9).	31/12/2013

Peersoft physical package description

Peersoft consists of a zip archive (Peersoftv1.5.zip) containing:

- One disk archive with the .do suffix (DOS 3.3 sector order) providing a complete Merlin 8 environment (DOS 3.3 version 2.47 by Glen Bredon from Roger Wagner Publishing) and full assembly source code to build Peersoft.

Filename(s) on disk	Purpose
PEERSOFTV15.S	Main source file for building Peersoft using Merlin
T.PEERINSTALL	Peersoft PUT file inclusing Peersoft installation stuff.
T.PEERLIST	Peersoft PUT file handling the LIST Applesoft instruction.
T.PEERINTEGARITH	Peersoft PUT file handling the integer arithmetic routines (handling the `+=`, `-=`, `*=` and `/=` for integer variables) as well as the loop variable increment whenever that loop variable is integer (`NEXT` statement)
CRECON.S	Assembly source file for handling CPU recognition upon Peersoft boot (see HELLO Applesoft program from other disk): object installs as a $0300 subroutine and thus helping launching the proper Peersoft executable file.
TCPRECON.S	Assembly source file for detecting a thunder clock peripheral card or a //gs clock chip: both can be used to measure time spent by Applesoft subroutines within the TF Applesoft program. Object file is loaded as a $0300 subroutine from TF BASIC program.
SMTRECON.S	Assembly source file for detecting a NoSlotClock device within an Apple][, //e or //c and making use of it.
TUTMC.S	Assembly source file for monitoring context switches as used by the TUTORIAL Applesoft program which serves as a demo for the co-routines features). Here, only a speaker toggle is applied thus routines are rather short.
HELLO, MERLIN, ASM.1, ASM.2, ED.16, ED, XREF, FORMATER, XREF A, KEYMAC.S, KEYMAC, T.MACRO LIBRARY, T.SEND MSG, T.FPMACROS, MON.65C02, SOURCEROR, LABELS.S, LABELS, PRINTFILER.S, PRINTFILER, T.ROCKWELL MACROS, T.PRDEC, T.OUTPUT, MAKE DUMP.S, EDMAC.S, EDMAC	Files originating from a Merlin 8 standard distribution (v2.47 DOS3.3 flavor)

- One disk archive with the .do suffix (DOS 3.3 sector order) providing a bootable DOS 3.3 image with the binary exe files from Peersoft (either machine code or Apple soft sample files).

File name on disk	Purpose
HELLO	Boot program displaying a menu and prompting the user to select a valid Perrsoft program version to load according to CPU detected
CRECON	Object file loaded by HELLO program and which purpose is to detect host CPU flavour
PEERSOFTV15_6502, PEERSOFTV15_65C02, PEERSOFTV15_65802	One binary executable file per CPU, each file results from assembly from PEEROFTV14.S source files with different setting for the KOPT and KOPT16 macros
TF	Applesoft sample program trying to illustrate the features currently included within Peersoft.
TCPRECON	Object file loaded by TF program which deals with peripheral clock/chip detection and time elapsed measurements (Thunder clock and //gs internal battery/clock chip).
SMTRECON	Object file loaded by TF program which deals with peripheral clock/chip detection and time elapsed measurements (NoSlotClock device chip).
TUTORIAL	Applesoft program file playing with the Peersoft mechanism of co-routines in some unusual ways.
TUTMC	Auxiliary machine code routines (loaded at address $0300). Currently only serves to toggle the speaker as a coroutine is swaped-in/swaped-out.

- Peersoft documentation in the form of a PDF file you are currently reading

How to transfer the two disk archives to real 5’1/4 disks on an Apple // hardware

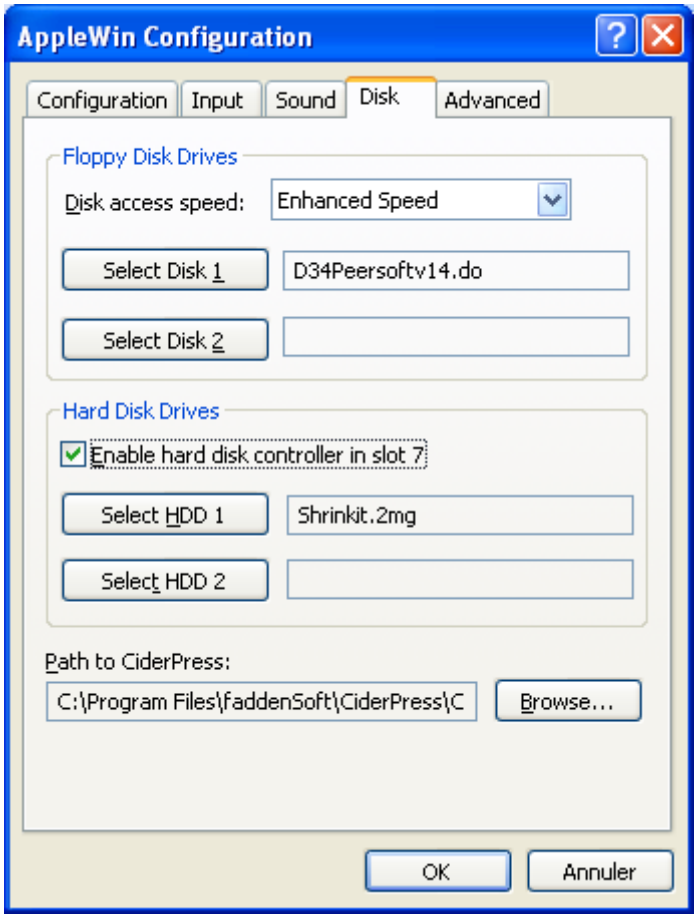
In order to transfer the content of both disks to a native hardware Apple // computer. We will use the components below:

- A pre-formatted disk image (with ProDOS2.0.3 and NuFX Shrink/Unshrink system files); this disk image will grab the DOS 3.3 disk images and put them into a ProDOS 8 archive file. In case you do cannot put your hand on such disk image, you can get the one from my site (URL is http://bgilon.free.fr/apple2/ShrinkIt.2mg).
- An emulator for running the NuFX Shrinkit program on your modern computer. To illustrate this, I am using the AppleWin 1.22 Win32 emulator.
- The CiderPress Win32 program in order to put the NuFX archive files onto a CFFA compact memory.
- The CFFA Compact Flash memory disk drive for Apple //e or //gs (mine is CFFA 2.0, but a newer version has since been released with the ability to directly read .po and .do disk images).

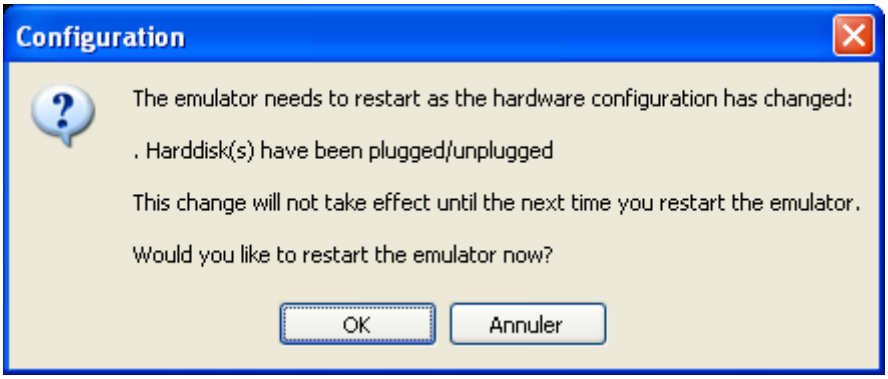
Other paths are available for performing the same tasks (dealing with serial communication interfaces between a “modern” computer and the Apple //).

Configuration of Apple Win 1.22

After having downloaded the disk image containing a bootable ProDOS 8 and the ShrinkIt system file, open the Configuration window by pressing the F8 function key.

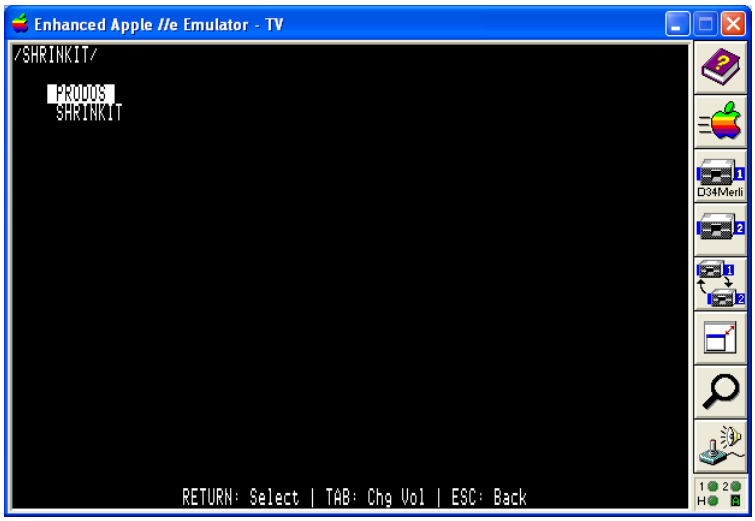


Check the option box labeled “Enable hard disk controller in slot 7” and click on the Select HDD 1 action button. A “choose file” dialog box would open. Navigate to your download directory and select the file. Click OK. A message box could then pop up advising you that AppleWin will reboot due to change in connected bootable devices configuration.

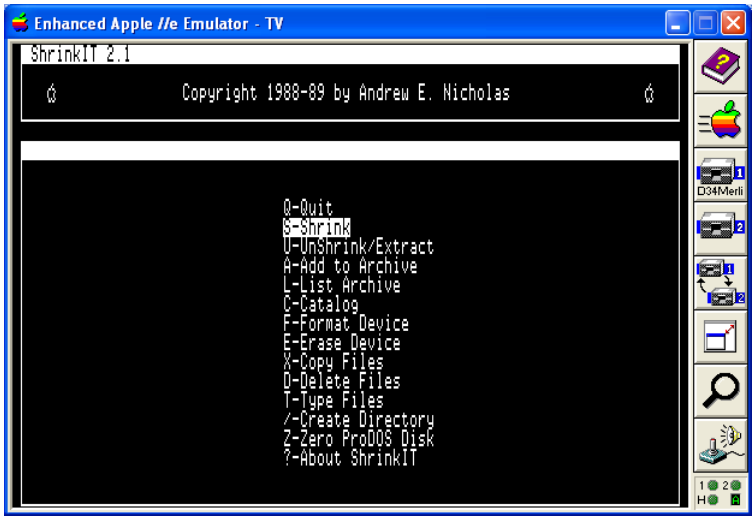

Configuration
The emulator needs to restart as the hardware configuration has changed:
. Harddisk(s) have been plugged/unplugged
This change will not take effect until the next time you restart the emulator.
Would you like to restart the emulator now?
OK
Annuler

Operation of the Shrinking procedure

Once the emulator has restarted, then the screen below should pop up.

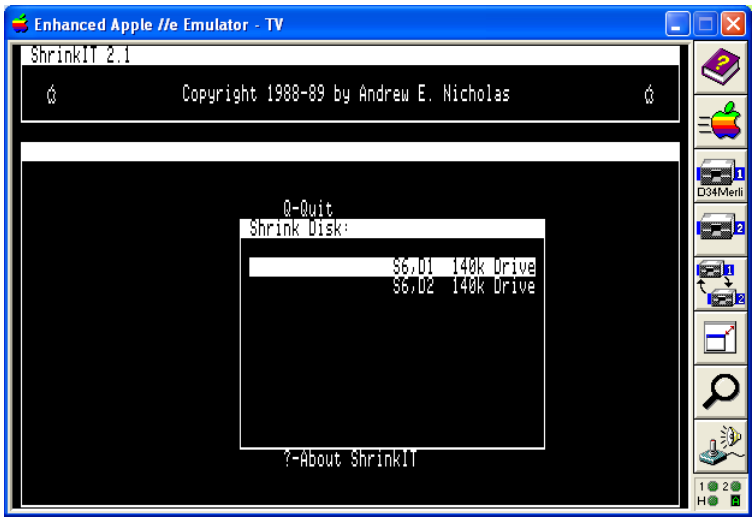


Select the SHRINKIT option, press Return and the screen below should pop up.

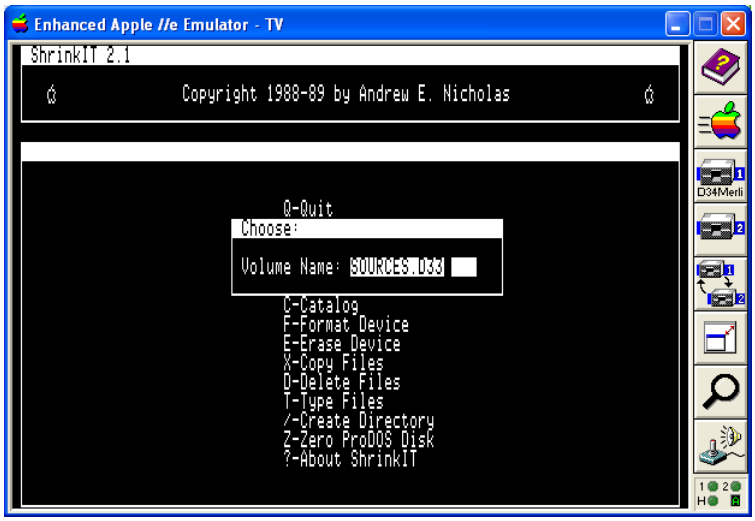


Drag and drop the D33Merlin – Peersoftv15.do disk image icon on the drive 1 box with the panel at the right side of the window. And select the Shrink option.

Select the Shrink “Disk” option

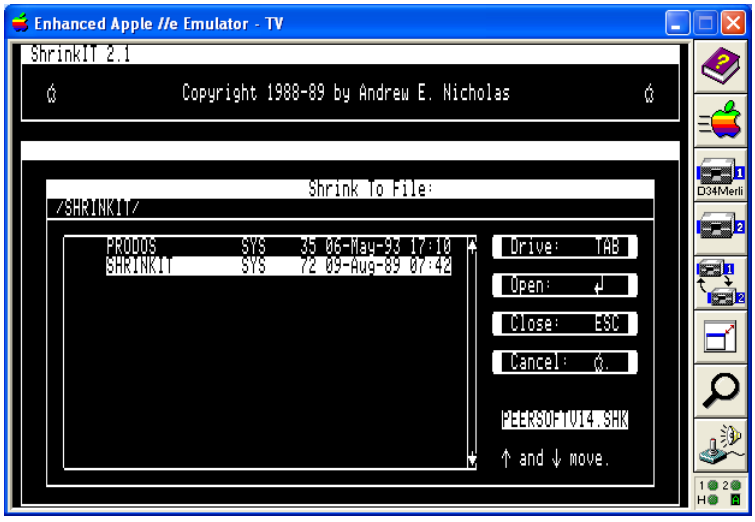


Select the “Shrink Disk on S6, D1 140k Drive” option..

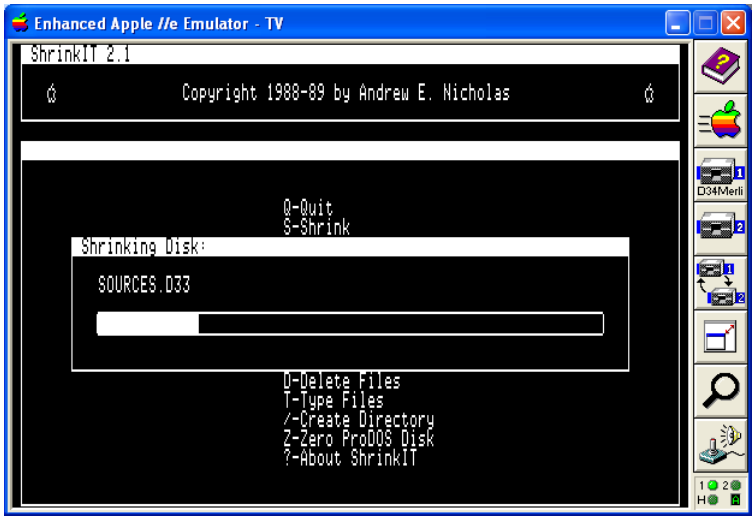


Enter the label for this backup within the archive. Here “SOURCES.D33”.

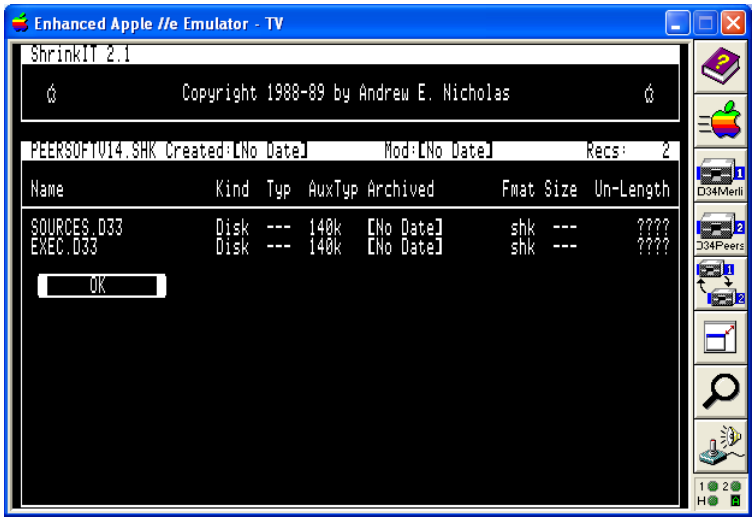
Now enter the name of the ProDOS file which will contain the backup you are about to initiate.



Here I have entered the filename PEERSOFTV15.SHK. Once the RETURN key has been pressed, the progression bar for the shrinking advises you of the… progress so far.



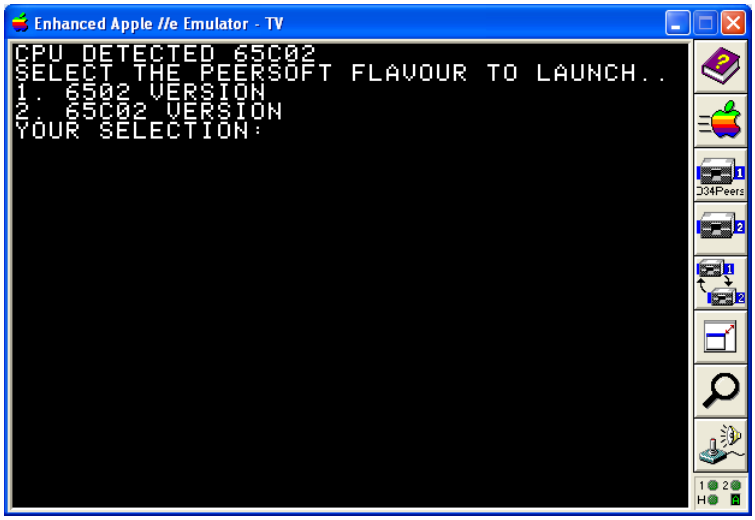
The next step would be to reiterate the procedure above for the D34Peersoftv14.do disk image. Checking that everything is OK at the end of this step can be done by listing the content of the archive which is an option available from the main menu.


Enhanced Apple //e Emulator - TV
ShrinkIT 2.1
Copyright 1988-89 by Andrew E. Nicholas
PEERSOFTV14.SHK Created:[No Date] Mod:[No Date] Recs: 2
Name Kind Typ AuxTyp Archived Fmat Size Un-Length
SOURCES.D33 Disk --- 140k [No Date] shk --- ????
EXEC.D33 Disk --- 140k [No Date] shk --- ????
OK

Peersoft user manual

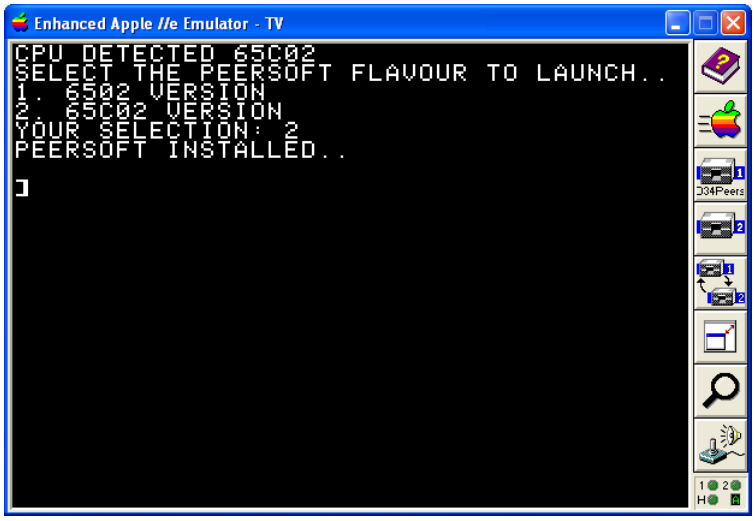
Peersoft executable files come within the D34Peersoft14 (.do archive or real 5’1/4 disk depending on whether you have an emulator or a real hardware on hand.

The relevant disk image is DOS 3.3 bootable, insert it in drive 6 slot 1 and reboot your emulator/computer. The screen appearing should be similar to the one below.



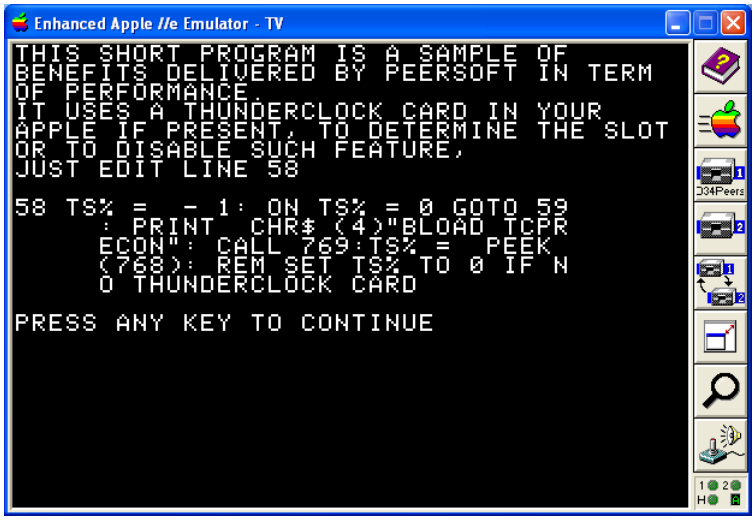
Depending upon the CPU detected on your host environment, more or less choices could be available. At this prompt, you can opt to bypass the Peersoft installation by using the usual <CTRL><C> keystroke. But for the time being, suppose that you selected option 2 to install the Peersoft version which can benefit from the richer instruction set of the 65C02 CPU.

The mention that Peersoft has been installed pops up.

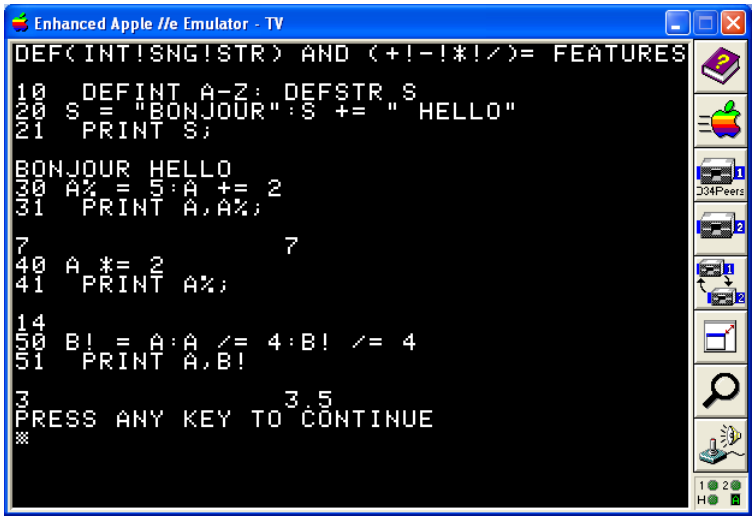


There is an Applesoft named TF which allowed the user to check that the latest build Peersoft showed no regression. In addition, it shows up every implemented feature to interested parties (either programmers themselves or end users).

Just issue the `RUN TF` command from the "`]`" prompt. This leads to screen below. The TF program can make use of the Apple //gs clock chip or of a Thunderclock peripheral card (as supported by the Virtual][emulator under Mac OS X) to measure time elapsed.


Enhanced Apple //e Emulator - TV
THIS SHORT PROGRAM IS A SAMPLE OF
BENEFITS DELIVERED BY PEERSOFT IN TERM
OF PERFORMANCE.
IT USES A THUNDERCLOCK CARD IN YOUR
APPLE IF PRESENT, TO DETERMINE THE SLOT
OR TO DISABLE SUCH FEATURE.
JUST EDIT LINE 58
58 TS% = - 1: ON TS% = 0 GOTO 59
: PRINT CHR$ (4)"BLOAD TCPR
ECON": CALL 769:TS% = PEEK
(768): REM SET TS% TO 0 IF N
O THUNDERCLOCK CARD
PRESS ANY KEY TO CONTINUE

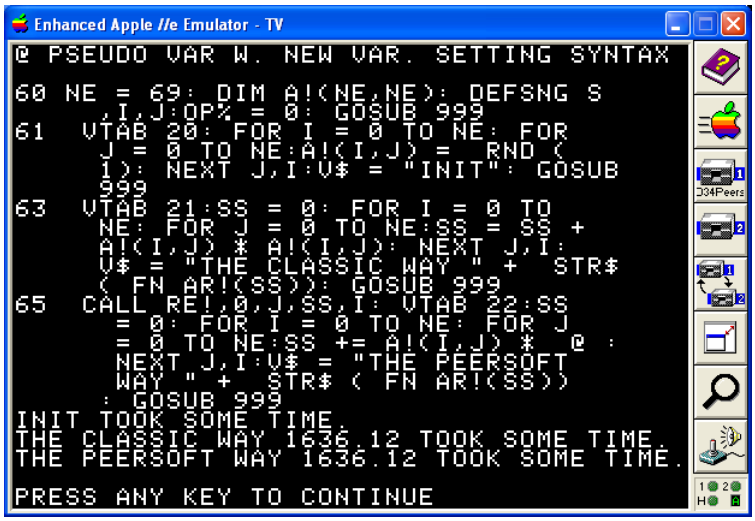
Just press any key on your keyboard to proceed…


Enhanced Apple //e Emulator - TV
DEF(INT!SNG!STR) AND (+!-!*!/)= FEATURES
10 DEFINT A-Z: DEFSTR S
20 S = "BONJOUR":S += " HELLO"
21 PRINT S;
BONJOUR HELLO
30 A% = 5:A += 2
31 PRINT A,A%;
7 7
40 A *= 2
41 PRINT A%;
14
50 B! = A:A /= 4:B! /= 4
51 PRINT A,B!
7 3.5
PRESS ANY KEY TO CONTINUE

This screen shows some new features available by using Peersoft as

- A new way to concatenate strings;
- Default typing for Applesoft variables (using the DEFSTR, DEFINT and DEFSNG statements);
- Some new syntax schemes for altering values of variables.

Just press any key on your keyboard to proceed…

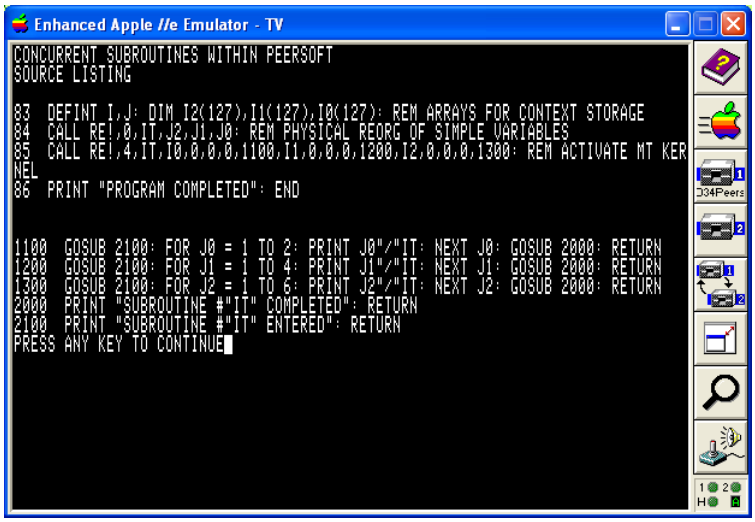

Enhanced Apple //e Emulator - TV
@ PSEUDO VAR W. NEW VAR. SETTING SYNTAX
60 NE = 69: DIM A!(NE,NE): DEFSNG S
,I,J:OP% = 0: GOSUB 999
61 VTAB 20: FOR I = 0 TO NE: FOR
J = 0 TO NE:A!(I,J) = RND (
1): NEXT J,I:V$ = "INIT": GOSUB
999
63 VTAB 21:SS = 0: FOR I = 0 TO
NE: FOR J = 0 TO NE:SS = SS +
A!(I,J) * A!(I,J): NEXT J,I:
V$ = "THE CLASSIC WAY " + STR$
(FN AR!(SS)): GOSUB 999
65 CALL RE!,0,J,SS,I: VTAB 22:SS
= 0: FOR I = 0 TO NE: FOR J
= 0 TO NE:SS += A!(I,J) * @ :
NEXT J,I:V$ = "THE PEERSOFT
WAY " + STR$ (FN AR!(SS))
: GOSUB 999
INIT TOOK SOME TIME.
THE CLASSIC WAY 1636.12 TOOK SOME TIME.
THE PEERSOFT WAY 1636.12 TOOK SOME TIME.
PRESS ANY KEY TO CONTINUE

Here no clock was detected (no support from the AppleWin emulator is provided) thus the only mention the program delivers is meaningless and the user should have a convenient external medium for time measurement this time.

The features that are showed within this screen are:

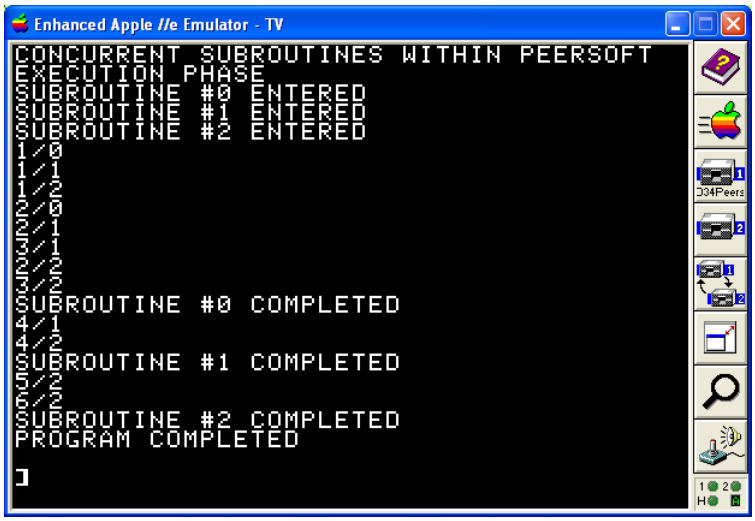
- Use of a utility routine to physically reorganize the simple variables memory area so that the variables "`J`", "`SS`" and "`I`" are looked up first from then on;
- Use of the "`@`" pseudo variable in order to avoid additional references (lookups) to simple variables "`I`", "`J`" and to array "`A!`" and relevant computations.

Just press any key to go to next screen.


Enhanced Apple //e Emulator - TV
CONCURRENT SUBROUTINES WITHIN PEERSOFT
SOURCE LISTING
83 DEFINT I,J: DIM I2(127),I1(127),I0(127): REM ARRAYS FOR CONTEXT STORAGE
84 CALL RE!,0,I%,J2,J1,J0: REM PHYSICAL REORG OF SIMPLE VARIABLES
85 CALL RE!,4,I%,I0,0,0,0,1100,I1,0,0,0,1200,I2,0,0,0,1300: REM ACTIVATE MT KER
NEL
86 PRINT "PROGRAM COMPLETED": END
1100 GOSUB 2100: FOR J0 = 1 TO 2: PRINT J0"/"I%: NEXT J0: GOSUB 2000: RETURN
1200 GOSUB 2100: FOR J1 = 1 TO 4: PRINT J1"/"I%: NEXT J1: GOSUB 2000: RETURN
1300 GOSUB 2100: FOR J2 = 1 TO 6: PRINT J2"/"I%: NEXT J2: GOSUB 2000: RETURN
2000 PRINT "SUBROUTINE #"I% "COMPLETED": RETURN
2100 PRINT "SUBROUTINE #"I% "ENTERED": RETURN
PRESS ANY KEY TO CONTINUE

Here is a minimalist segment of code to illustrate the use of co routines within Peersoft (an innovative feature indeed). The first few lines listed set up the environment. And the latter lines form the body of the co routines and subroutines called from within the co routines.

To show what is displayed on the screen resulting from running the activation, just press a key as usual.



Every co routine is entered and completed and the allocation of CPU to each thread follows a round robin model till all subroutines complete, thus triggering the end of program. The co routine feature is thoroughly described in a subsequent chapter.

Peersoft reference manual

Variable default typing

DEFINT A, I-N,Z

To specify the scope of variables involved by every DEF<type> instruction, you just have to list first characters either alone or as part of a range. In the sample above, every variable (simple or array), which first character is “A”,”I”,”J”,”K”,”L”,”M”,”N” or “Z” will have a default type of integer.

The involved statements are DEFINT, DEFSNG, DEFSTR, DEFBYTE, DEFLNG24 and DEFLNG32 (the three latter being added with release of version 1.5).

DEF<type> statements can appear anywhere within a program text and be run at anytime within the program flow.

At program start or when a RUN/CLEAR Applesoft statement is run within the program flow, then all variables inherit the Floating point type default.

However, an explicit type specifier (“%”, “$” or the new “!” used for floating point variables or the new “.”, “]” and “[” for alternate integer types) as a variable name’s suffix overrides its default type currently defined.

Thus the statement sequence `CLEAR : DEFINT I:I = 1: PRINT I!` will print 0 on screen.

New syntax scheme for altering variables values

`A += 3`

Peersoft will simplify variable value alterations by providing a new syntax scheme. The new `A += 3` being a shortcut for `A = A + 3`.

All four basic operations can be part of the new syntax scheme.

For instance A -= 3 is a shortcut for `A = A - 3` and `B /= 4` is a shortcut for `B = B / 4`

The new syntax scheme can be included in every context where a variable has its value set within program text. This includes the FOR/NEXT loop construct.

`FOR I += 5 TO 10` is a shortcut for `FOR I = I + 5 TO 10`

Whenever the variable type is integer, then arithmetic operation applied is of the integer kind too. Also the += syntax scheme is also valid for string concatenation whenever the variable is a character string.

`S$ = "BONJOUR": S$ += " HELLO": PRINT S$` will print BONJOUR HELLO onto the output display.

@ Pseudo variable

Having written quite a number of applications myself and studied the code from other authors as well, I’ve found out that one pattern that emerge quite often is the use of the same sub expression/variable multiple times within an expression. Sometimes, there is a cost in term of

performance to lookup some sub expression/variable (particularly when dealing with multi dimensions array variables). So the idea of implementing the `@` pseudo variable was born.

Anytime an expression is evaluated, then the Applesoft interpreter will use some constant locations within page zero as main and auxiliary accumulators, large enough to contain an integer, a floating point value, or a string descriptor. The `@` pseudo variable is the simplest in its processing code. All it does is a RTS (actually it's a bit more than that but only by a small amount: cf. source code for further details ;-), this would imply that the returned value will come unchanged from what it was during the last "factor" evaluation.

Beyond the sample code showing up in the previous section : "Peersoft user manual", the `@` could also refer to any content of any type.

`PRINT RIGHT$( A$, LEN( @ ) - 4)` will print the "`A$`" current value with its first 4 characters removed.

Use of the `@` pseudo variable as the first term evaluated within an expression thus possibly referencing the result from a previous statement expression is not recommended, especially from within a co routine (cf. section named "Co routines within Peersoft" for details upon what a co routine is in author's mind).

Integer variables as loop variables within FOR/NEXT loops

I was worried that Integer variables be banned from being used as loop variables within FOR/NEXT loop constructs.

If you try the statement below under plain vanilla Applesoft interpreter, all that is returned is a "`?SYNTAX ERROR`" message.

```
FOR I% = 1 TO 10: PRINT I%: NEXT I%
```

I believed that the use of integer arithmetic for processing the increment and test for final value as part of the NEXT statement processing would greatly offer benefits in performance terms.

Hélas (in French in the text), by the time the loop variable appears in the loop body, then all benefits disappear because handling of integer variables is much more costly than of floating point variables just by the fact that the retrieved integer value needs to be converted to floating point whatever the context.

OK now, with Peersoft installed, you can have integer variables as loop variables, but I wouldn't tell you more about it… until September this year (cf. section "Peersoft roadmap" for further details on future developments).

Ah yes, be advised that Applesoft is bugged in its integer variable handling too. Have you ever tried to issue a `A% = - 32768` only to get bounced with a `?ILLEGAL QUANTITY ERROR` message?

Users curious about this state of things could study the excellent Web resource already mentioned in this document (http://www.txbobsc.com/scsc/scdocumentor/). Suffice to say that `A%= - 32767.5` works well and provides the same expected result.

Another limitation to warn the reader about is that the final value of such loop (using integer variables) cannot be 32767 (which is the algebraic highest possible value an integer variable can be bound to). This is because, as the last iteration (the loop variable being equal to the final value) completes, the first operation the NEXT statement does is to increment the loop variable's value (here 32767) with the STEP value (default 1), this add operation causes an overflow within the 6502 and thus the overflow exception is raised to the Applesoft environment, itself raising an "?OVERFLOW ERROR" for the Applesoft program.

The same limitation exists for a final value as -32768 when the STEP value is -1.

Some Applesoft statements processing bugfixes

Every bug fix provided here is an obvious code update to bugs raised as part of the Web resource already mentioned (http://www.txbobsc.com/scsc/scdocumentor/).

ONERR statement

The current Applesoft implementation for the ONERR statement processing erroneously skip the whole physical line after processing instead of just up to next "end of instruction" marker.

This is fixed within this Peersoft release.

RETURN and POP statements

Co routines within Peersoft

A word of advice: this section describes the working of co routines features within the 1.4 release of Peersoft. Future releases might expose other API to the external entities, in case this interface evolves or others appear, then this section will be updated accordingly within the document you are currently reading.

From now on, an Applesoft program should be considered as a sequence of consecutives phases: some with active co routines (flows of control running in // and on different parts of the program text) and inactive co routines (when a unique flow of control exists and determines the program behavior).

Except when you previously installed another "multi-tasking" environment in your Apple 2, then every Applesoft application has to be considered as a purely sequential unit of flow on time before the adoption of Peersoft in its current incarnation.

Peersoft provides two ways of doing for both transitions (one for from "purely sequential" to "active co routines" and the other for the other way round).

Activating the co routines (simplistic sample)

For activating the MT (short for "multi threading" kernel) and the co routines which come along, then a unique way of doing this is given below (minimal sample):

```
5 DIM I0%(127),I1%(127), I2%(127)
10 RE! = PEEK( 40160) + 256 * PEEK( 40161)
20  PRINT "ACTIVE CO ROUTINES PHASE ABOUT TO BEGIN ON LINES 1000,
2000 AND 3000"
```

```
30  CALL RE!,4,IT%,I0%,0,0,0,1000,I1%,0,0,0,2000,I2%,0,0,0,3000
40  PRINT "ACTIVE CO ROUTINES PHASE ENDED": END
1000  GOSUB 5000: FOR J0% = 1 TO 2: PRINT J0%;"/";IT%: NEXT
1010  GOSUB 5010: RETURN
2000  GOSUB 5000: FOR J1% = 1 TO 4: PRINT J1%;"/";IT%: NEXT
2001  GOSUB 5010: RETURN
3000  GOSUB 5000: FOR J2% = 1 TO 6: PRINT J2%;"/";IT%: NEXT
3001  GOSUB 5010: RETURN
5000  PRINT "CO ROUTINE #";IT%;" ENTERED": RETURN
5010  PRINT "CO ROUTINE #";IT%;" ABOUT TO QUIT": RETURN
```

Address 40160 contains a pointer to the general utility routine within Peersoft.

The arguments are described in the table below:

Table 2: Arguments for activating the MT kernel

Argument		Description
4		Reason code meaning: I would like to activate the MT kernel with co routines defined by following parm values.
IT%		It is the name of the Applesoft variable (must be simple integer type variable) which will hold the current thread index value from 0 to *NumCoRoutines – 1*. Peersoft updates this value upon every context switch.
1st co routine	I0%	Name of the integer type array which will contain the context storage for the 1st co routine, structure of this array is given in a section below.
	0	This parameter defines whether the co routine has a private error handling routine of its own. This parameter should be considered as a bit string here where, for our purpose only the two lsb interest use. Three values are possible here: 0: implies that no error handling at all while the co routine is the one run by CPU. That means that no segment exists in the context dealing specifically with the error handling, making its size smaller and its store and retrieval faster. Whenever the context is restored, a zero is stored in the ERRFLG flag page zero location. 1: Private error handling which instructs Peersoft to cater for dedicated error handling segment within stored context for this co routine. The co routine should however, execute an ONERR GOTO nnn instruction in its own flow of control. 2: The co routine relies on the status of the "global environment" (ie error handling status as the CALL RE!, 4,… is run), a context segment for dealing with error handling is created iif the ERRFLG (page zero location $D8 meaning an ONERR handler is active) is set upon the CALL RE!,4,… is processed by Peersoft. The role of other bits (b2b7 from the value are described in a subsequent section).
	0	This parameter is the address of a machine language subroutine (ending with a RTS instruction) called whenever the co routine is about to be active (gain the 6502 CPU). The sub routine must not change any register value (cf. Push and Pull 6502 instructions)
	0	This parameter is the address of a machine language subroutine (ending with a RTS instruction) called whenever the co routine is

		about to release control and the corresponding context be stored in the context storage area (see array I0% description above). The sub routine must not change any register value (cf. Push and Pull 6502 instructions)
	1000	This is the co routine starting Applesoft BASIC line number. Consider that, internally, the CALL RE!, 4… does a GOSUB to this line number upon co routine activation.

Arguments descriptions for 2nd and 3rd co routines are similar in their description as the ones for the 1st co routine's arguments. Up to 8 co routines can be active at the same time.

Activating the co routines (Not so simplistic approach)

Now suppose that a particular co routine needs to have a dedicated environment for text cursor positioning.

The context segment representative of text cursor positioning could be summarized within the table below:

WNDLFT, WNDWDTH, WNDTOP, WNDHGHT for text window setting on the display screen and

CH, CV, BASL, BASL+1 for cursor location within the window.

Deactivating the co routines

Beyond the natural and normal way of returning to a single flow for the Applesoft application (let every co routine return to the statement following the `CALL RE!,4,…` instruction (by using a combination of `RETURN`/`POP` statements themselves).

The fastest way is based upon a new reason code for the Peersoft general utility: `CALL RE!,5` is the instruction to insert within the code of a co routine (including its error handling procedure, dedicated or shared). Such statement **must** be run as the MTK is active.

Peersoft data structures and hints for performing usual tasks from Applesoft programs relating to co routines

For reference by assembly language programmers, here is the structure of the Peersoft global page and of every integer type array variable used by Peersoft for context storage purposes.

Table 3 : Peersoft global page

Address (decimal)	Address (hexadecimal)	Description
40159	$9CDF	A call to this address will branch to the Peersoft "general utility" routine already described in a previous section. An alternate way is to get the vector stored at (40159+1, 40159+2) and calling it directly (cf. sample Applesoft above)
40158	$9CDE	Peersoft version byte: currently a $15 value is stored at this location (meaning 1.5)
40157	$9CDD	Number of instructions between two context switches (default to 10, setup whenever Peersoft is loaded from disk).
40156	$9CDC	Bit 7 set iif the MT kernel is active. A call to CALL RE!,4,… will set it up. This flag is reset whenever the MT kernel is terminated, usually as the last co routine returns to the global environment.

40155	$9CDB	Number of ticks that the currently running co routine will last before next context switch. At every context switch, Peersoft copies the $9CDD slot into this slot, upon running an Applesoft instruction, the context switch occurs only if the value from this slot, decremented by one, reaches zero.
40154	$9CDA	Bit 7 set if context switch temporarily inhibited while a critical section of code is run by the current co routine.

What happens when the co routines are established?

A `GOSUB` stack frame is created in the stack segment of every co routine's context. This GOSUB frame indicates that the return points to just after the `CALL RE!,4,…` Applesoft statement. Obviously, the stack pointer for every established co routine is decremented by the frame size (5 bytes including the GOSUB token).

Peersoft marks a co routine as being completed when, as this co routine is being run by the CPU, the current stack pointer reaches the original stack pointer value taken as the CALL RE!, 4.. Applesoft statement was parsed.

Hints and tips

How to release control to other threads from the current co routine?

Just use the `POKE` statement `POKE 40155,1` just before the location where you want to release control. While parsing the next statement, Peersoft will decrement this value to 0 and thus a context switch will be triggered (saving the current context, and restoring the next active co routine declared within the kernel. Be advised that this could be the same co routine as the current one in case no other is still active.

How to temporarily disable the context switch?

While a critical section of Applesoft code is being run within the current co routine, no context switch should occur in order to let this section of code appear as being atomic. A simple way to fit this requirement is to insert a `POKE 40154, 128` statement at the beginning of your critical section code. In order to reinstate switches for giving a chance for other co routine to flow normally, then use the `POKE 40154, 0` statement at the conclusion of this section.

Also, as the decrement operation is not processed while the switching is inhibited, it is a good idea to insert a `POKE 40155, low_value` just in the vicinity of the former `POKE` statement (as I did in my tutorial example Applesoft programs).

Having private variable sets (no collision between co routines)

The current solution I propose is to get arrays of variables with at least one dimension indexed by the context index value. In the tutorials from the disk, I used two arrays (`XH()` and `XV()`) to store cursor data (line and column where cursor lies in two dedicated integer arrays) and all PRINT statements or cursor position setting statements being run in critical sections of code.

Committing suicide or assassination (of other threads)

Beyond the usual way to mark a thread as completed (i.e. using RETURN or POP instructions in order for the stack pointer to reach its initial value), an alternate and more intrusive way would be to force a specific byte from Peersoft memory to $FF value, thus Peersoft will consider the relevant co routine as completed. Here is the code segment which performs just that action.

```
AD =  PEEK( 40152) + 256 * PEEK( 40153): POKE AD + 8 + IT%,255
```

Where IT% being the current context index implies suicide and IT% being unequal to current context index (but still between 0 and 7) meaning assassination.

Structure of the context storage

Every context is stored within a dedicated integer type array variable (one dimension) which layout is described in the table below

Table 4 : Context Storage layout

Offset	Page zero	Description
Header for housekeeping by Applesoft		
0 and 1	N/A	Name of the array (two bytes)
2 and 3		Offset from the beginning of this array to next array variable or to end of memory area
4		Number of dimensions (must be 1 for Peersoft usage).
5 and 6		Value of first (and last) dimension
Constant segment (general use)		
7	N/A	Offset to stack segment (always populated)
8	N/A	Operation mode for context. B0b1 provides an indication whether the local error handling is in use or not. In case local error handling is in use, whether the global environment is used for such context or not; B2b7 provides options for additional context switch operations. The one being shown within the tutorial is the display cursor backup/restore operations.
Constant segment for monitoring context switches		
9 and 10	N/A	Address of machine language routine to be called whenever the co routine is paged in. This routine must not alter register values from the calling environment (unless pushed on stack) and must return with a RTS (after possible Pull from stack instructions). High byte is $FF if no routine registered.
11 and 12	N/A	Address of machine language routine to be called whenever the co routine is paged out. This routine must not alter register values from the calling environment (unless pushed on stack) and must return with a RTS (after possible Pull from stack instructions). High byte is $FF if no routine registered.
Core segment (always populated)		
13	REMSTK ($D8)	Current stack pointer for this pointer (only byte at offset 8 is meaningful)
14 and 15	CURLIN, CURLIN+1	Current Applesoft line # for the co routine
16 and 17	TXTPTR ($B8), TXTPTR+1	Current text pointer within program text for the co routine
18 and 19	OLDTEXT, OLDTEXT+1	Text pointer of last instruction parsed by interpreter exec loop
Local Error handling segment (optional: see value at offset 8)		

20 and 21	TXTPSV ($F4), TXTPSV+1	Points to the first character of line # as ONERR GOTO statement is parsed.
22 and 23	CURLSV ($F6), CURLSV+1	Line # where the ONERR GOTO is located
24	ERRNUM ($DE)	Error # when an error occurs
25	ERRSTK	Stack pointer as the error occurs (so that RESUME could branch back to the faulty statement)
26 and 27	ERRLIN ($DA), ERRLIN+1	Applesoft line # where the error occurred (so that RESUME could branch back to the faulty statement)
28 and 29	ERRPOS ($DC), ERRPOS+1	TXTPTR pointer of the statement raising the error.
30	ERRFLG ($D8)	Only bit 7 is meaningful here.
Stack segment (variable size)		
<ValueAt Offset 7> and above	N/A in page zero: within hardware page 1	From private stack pointer to global environment stack pointer value

Peersoft co routines tutorial

Within the Applesoft program listing below,

- The arrays `I0`, `I1` and `I2` serve as context storage areas useful for switching between co routines;
- The arrays `XH` and `XV` serve as memory place where to store screen cursor locations (horizontal for `XH` and vertical for `XV`) for every co routine implemented here (the number of them being 3).

- Variable XC serve as an indicator that the user issued a Ctrl-C keystroke while the program was running. Therefore it is set to a non zero value at line 2901 (part of the shared general error handling routine beginning at line 4000 (see ONERR statement at line 2 and part also of the dedicated (i.e. private) error handling for co routine #1 beginning at line 2900);
- RE holds the address where to call the Peersoft general utility routine with appropriate parameters.

```
1 CLEAR : DEFINT I-N,X: DIM I0(127),I1(127),I2(127),XH(2),XV(2)
2 PRINT CHR$ (4)"PR#0": TEXT : HOME :XC = 0: ONERR GOTO 4000
3 PRINT CHR$ (4)"BLOAD TUTMC": VTAB 1: HTAB 15: PRINT "TUTORIAL 2"
4 XH(0) = 1:XV(0) = 2:XH(1) = 1:XV(1) = 21:XH(2) = 1:XV(2) = 6: DEF FN
DR(A) =
PEEK (A) + 256 * PEEK (A + 1): DEF FN AR(CX) = INT (CX * 100) * .01
5 RE = FN DR(40160): POKE 40157,4: REM # OF APPLESOFT INSTRUCTIONS RUN
BETWEEN TWO SWITCHES
6 CALL RE,4,IT,I0,2,0,0,1000,I1,1,768,774,2000,I2,2,774,768,3000
7 VTAB 1: HTAB 1: PRINT "PROGRAM ENDED, PRESS ANY KEY";: GET A$: HOME : END
999 REM FIRST COROUTINE: MONITOR EVERY CONTEXT INCLUDING ITSELF
1000 AD = FN DR(40152):OF = 0:NT = 0:SO = PEEK (AD + 17):SL = 0:LX = - 1:
GOS
UB 5010: PRINT " RUNNING TASKS STATUS (";SO"/";: GOSUB 5000:XH = XH(IT):XV
= XV(IT)
1002 FOR JT = 0 TO 7: ON PEEK (AD + 8 + JT) < 255 GOTO 1003:NT = JT - 1:JT
= 7
1003 NEXT JT: FOR J0 = 0 TO 1 STEP 0: GOSUB 1100:JF = 1
1004 FOR JT = 0 TO NT: GOSUB 1200: NEXT JT
1005 J0 = JF: NEXT J0: RETURN
1099 REM
1100 ON PEEK (40157) = LX GOTO 1102: POKE 40154,128: HTAB XH: VTAB XV:LX =
PE
EK (40157)
1101 PRINT LX;")";: CALL - 868: POKE 40155,1: POKE 40154,0
1102 RETURN
1199 REM PRINT A CONTEXT CONTENT (JT)
1200 IF PEEK (AD + JT + 8) < 255 AND JT < > IT THEN JF = 0
1201 OF = PEEK (AD + JT + 8) * 256 + PEEK (AD + JT):XV(IT) = 3 + JT:XH(IT)
= 1
: GOSUB 5010: CALL 777,OF,JT: GOSUB 5000
1202 RETURN
1999 REM SECOND CONTEXT: PROCESS SOME KEYBOARD INPUT FROM USER
2000 BS$ = CHR$ (8):CU$ = CHR$ (127) + BS$: POKE 49168,0: ONERR GOTO 2900
2001 GOSUB 5010: PRINT SPC( 6);"DIVISION EXEMPLE": GOSUB 5000:LY = XV(IT):
FOR
J1 = 0 TO 1 STEP 0
2002 XH(IT) = 1:XV(IT) = LY: GOSUB 5010: CALL - 958: PRINT "ENTER
NUMERATOR: "C
U$;: GOSUB 2801: ON M$ = "" GOTO 2004:VN = VAL (M$)
2003 GOSUB 5010: PRINT "ENTER DIVISOR: "CU$;: GOSUB 2801: ON M$ < > "" GOTO
2005
2004 J1 = 1
2005 ON J1 = 1 GOTO 2007:VD = VAL (M$):VR = FN AR(VN / VD): GOSUB 5010:
PRINT
"RESULT: ";VR;" <RET> TO PROCEED"CU$;: GOSUB 2851: ON XC = 1 OR ES% = 1
GOTO 20
04: GOTO 2007
2006 POKE 40154,128: VTAB 24: HTAB 1: PRINT MO$;
2007 NEXT : RETURN
2800 REM INPUT SUBROUTINE
2801 GOSUB 5000:M$ = "":LM = 0:ES% = 0: FOR JS = 0 TO 1 STEP 0
2802 GOSUB 2861: ON ES% = 0 AND XC = 0 GOTO 2803:M$ = "":LM = 0: GOTO 2809
```

```
2803 ON JS = 1 GOTO 2809: ON A < > 8 OR LM = 0 GOTO 2804:LM -= 1:M$ = LEFT$
(
M$,LM + (LM = 0)): PRINT " "A$A$;CU$;: ON LM > 0 GOTO 2804:M$ = ""
2804 ON A < 31 GOTO 2809:LM += 1:M$ += A$: PRINT A$;CU$;
2809 GOSUB 5000: NEXT
2810 GOSUB 5010: CALL - 868: PRINT : GOSUB 5000: RETURN
2850 REM GET RETURN SUBROUTINE
2851 GOSUB 5000:ES% = 0: FOR JS = 0 TO 1 STEP 0
2852 GOSUB 2861:JS = (ES% = 1) OR (XC = 1) OR (A = 13): GOSUB 5000: NEXT :
GOSU
B 5010: CALL - 868: GOSUB 5000: RETURN
2860 REM GET KEYBOARD ENTRY
2861 ON PEEK (49152) > 127 OR XC = 1 GOTO 2862: POKE 40155,1: GOTO 2861
2862 GOSUB 5010: IF XC = 0 THEN GET A$:A = ASC (A$)
2863 ON XC = 0 GOTO 2864: PRINT "#ABORTED#!";:JS = 1
2864 ON A < > 27 GOTO 2865: PRINT "<ESCAPED>";:JS = 1:ES% = 1
2865 ON A < > 13 GOTO 2866:JS = 1
2866 RETURN
2900 ON PEEK (222) < > 255 GOTO 2902
2901 XC = 1:A$ = CHR$ (3):A = 3: PRINT CHR$ (7);: RESUME
2902 ON PEEK (222) < > 133 GOTO 2903:EL = FN DR(218): ON EL < > 2005 GOTO 2
903:MO$ = "DIVIDE BY ZERO ERROR":J1 = 1: CALL - 3288: GOTO 2006
2903 PRINT CHR$ (7);: GOTO 4003
2998 REM 3RD CONTEXT MAIN ROUTINE, JUST PRINT SOME STAR CHARACTERS
2998 REM 3RD CONTEXT MAIN ROUTINE, JUST PRINT SOME STAR CHARACTERS
2999 REM AS A BACKGROUND ACTIVITY
3000 FOR J2 = 0 TO 1 STEP 0:J2 = J1: GOSUB 3008
3001 PRINT "*";: GOSUB 5000: NEXT
3002 FOR J2 = 1 TO 4: GOSUB 3008: PRINT MID$ ("OVER",J2,1);: GOSUB 5000:
NEXT
: RETURN
3008 XV(IT) = INT ( RND (1) * 15) + 6:XH(IT) = INT ( RND (1) * 40) + 1:
GOSUB
5010: RETURN
4000 IF PEEK (40156) < 128 THEN VTAB 23: HTAB 1: CALL 771: END
4001 ON PEEK (222) = 255 GOTO 2901
4003 XH(IT) = 1:XV(IT) = 23: GOSUB 5010: CALL 771: GOSUB 5000: CALL RE,5
4998 REM STORE CURSOR POSITION INTO CONTEXT AND RELEASE CONTROL TO MT
4999 REM EXPECTS TO BE CALLED WHILE CONTEXT SWITCHES INHIBITED
5000 XH(IT) = PEEK (36) + 1:XV(IT) = PEEK (37) + 1: POKE 40155,1: POKE
40154,0
: RETURN
5009 REM INHIBIT CONTEXT SWITCH AND RESTORE CURSOR POSITION FROM STORED
CONTEXT
5010 POKE 40154,128: VTAB XV(IT): HTAB XH(IT): RETURN
```

Sommaire

An introduction to Peersoft 2

Peersoft roadmap 4

Peersoft physical package description 5

How to transfer the two disk archives to real 5’1/4 disks on an Apple // hardware 6

Configuration of Apple Win 1.22 7

Operation of the Shrinking procedure 7

Peersoft user manual 11

Peersoft reference manual 15

Variable default typing 15

New syntax scheme for altering variables values 15

@ Pseudo variable 15

Integer variables as loop variables within FOR/NEXT loops 16

Some Applesoft statements processing bugfixes 17

`ONERR` statement 17

`RETURN` and `POP` statements 17

Co routines within Peersoft 17

Activating the co routines 17

Deactivating the co routines 19

Peersoft data structures and hints for performing usual tasks from Applesoft programs relating to co routines 19

Peersoft co routines tutorial 22